

DEJA VIEW

Jonathan S. Sayles



IN WATCHING A RECENT "60 Minutes" program, I was impressed by a feature on dyslexia and an approach to curing the disease through the use of colored lenses. These lenses acted to filter out the confusing elements of reading (the seeming disorder of row upon row of phonetic symbols) and bring clarity and organization to the words on the page. Apparently, for some dyslexics, printed words appear as just so many unrelated characters. A coherent perspective of the raw elements of reading presented in text is required to bring the semantic meaning of seeming random patterns into focus. The customized, colored lenses provided a means of filtering extraneous information, allowing the dyslexics' eyes and mind to focus on the inherent meaning of the text.

(Continued On Page 16)

A PRIMER

What You Can Accomplish With Views.

1. Provide custom user/application data access by:
 - Selectively filtering rows of base table(s)
 - Selectively filtering columns of base table(s)
 - Creating custom tables composed of:
 - Joined tables
 - Statistical summary tables of aggregate functions
 - Transformed tables of scalar, arithmetic and other "pseudo-columns"
2. Provide logical program/data independence by allowing:
 - Tables to be split vertically
 - Tables to be split horizontally
 - Columns to be renamed
 - Columns to be deleted
3. Provide an extra measure of data security and integrity by:
 - Supporting row level access to table data
 - Supporting column level Selection of table data
 - Supporting hierarchical access to table data
 - Enforcing domain integrity
 - Pinpointing referential and other data integrity problems
4. Provide DB2 system and application efficiency controls by:
 - Forcing application and end-user access to base tables through optimized query paths for table joins and other complex SQL operations
 - Enforcing element-level data access on application programs
5. Facilitate application development by:
 - Maneuvering around compromises in the SQL language
 - Storing the business rules for DB2 applications in the DB2 System Catalog

(Continued From Page 15)

In much the same way as words on a page can seem like unordered information to dyslexics, DB2 databases - particularly ones containing denormalized tables - can overwhelm system users accustomed to a monolithic, logical view, organized according to their specific processing needs. Further, because relational technology "allow[s] us to assemble a database of facts without predetermining their physical relationships", we attempt to design databases which project a true corporate-level information model - A model that is not biased in favor of a specific application perspective of data relationships. This means that even given a well normalized logical design, a dichotomy often exists between the corporate data model and application specific views of the same information.

Applications personnel and business users want to focus on the particular data that interests them, and

on the particular tasks for which they're responsible. How can we maintain the integrity of sound relational design principles in the face of pressure from application developers and system users to provide a specific information framework?

One solution is to represent specific data relationships through the VIEW mechanism. Applied like the customized colored lenses over the data, VIEWS of DB2 databases can be used to simulate different semantic data models and support multiple application-specific venues of information without compromising the underlying conceptual design.

In fact, the view mechanism is even more powerful, useful and flexible than the colored lenses, facilitating DB2 Database and system administrator tasks in the areas of logical program/data independence, data security/integrity, system efficiency

and SQL facilitation.

Before we discuss how views contribute to this sleight-of-hand, let's take a brief look behind the scenes at views, specifically, what they are, and how they work.

OVERVIEW

Views can be best understood when they are described within the context of the three types of relational tables:

- Base Tables
- Results Tables
- Views

A base table is a permanent, defined DB2 object that can be thought of (for the purposes of this discussion) as physically storing data.

A results table is a temporary, undefined table created by the execution of a SELECT statement, that reflects data values in a base table(s) - at the moment the statement is executed.

(Continued On Page 17)

(Continued From Page 16)

A VIEW is a permanent, defined, stored SELECT statement that materializes into a results table when another SELECT statement that references that view is executed. Let's take a look at how each of the relational tables is built.

Base tables are defined with the CREATE table statement. Two of the tables we will be working with are defined in figure 1.

If I load a base table up with data and execute a SELECT statement, as shown in Figure 2, DB2 generates a temporary "results table" which reflects the data values physically stored in the base table at the exact moment the query is executed.

After I have finished looking at the results table, it goes away.

A VIEW is a permanent, stored SELECT statement. Views are defined with a name and generate results tables when referenced in SQL SELECT statements. Views are used to extend or filter a users' perception of data in a DB2 database by invoking the SQL operations they are composed of. An example of a view definition is shown in Figure 3.

The Figure 3 statement defines a view called YOUNG_EMPLOYEES. The view definition (the view name, along with the actual SELECT-FROM-WHERE text itself) is stored in the DB2 system catalog (SYSIBM.SYSVIEWS). Several other entries are also made in the system catalog regarding ownership of the view (SYSIBM.SYSTABAUTH), view/base table object dependencies (SYSIBM.SYSVIEWDEP), and internal view representation and storage (SYSIBM.SYSVTREE, AND SYSIBM.SYSVLTREE).

With the creation of a view, a SELECT statement that can define different data relationships among the elements in a DB2 table becomes a permanent named object, available to users by referencing the view name in the FROM clause of their queries. Understand that the particular rows from DB2 tables that would satisfy the view definition are not duplicated in any way. Those rows are extracted by DB2 during the execution of a SELECT statement against the view as shown in Figure 4.

What's more, DB2 allows you to manipulate views almost as easily as would manipulate the elements of the EMP base table, applying additional SELECT-FROM-WHERE criteria to it. How DB2 combines a SELECT statement against a view with the underlying view defi-

(Continued On Page 18)

Figure 1. Statements To Define Base Tables:

```
CREATE TABLE EMP (
  NBR                INTEGER
  NOT NULL.
  LNAME              CHAR(12) NOT NULL,
  FNAME              CHAR(06) NOT NULL WITH DEFAULT,
  DOB                DATE,
  HIREDTE            DATE    NOT NULL WITH DEFAULT,
  DEPT               CHAR(03),
  JOB                CHAR(04),
  PERF              SMALLINT,
  PROJ              CHAR(02) )
IN DATABASE SYSGROUP;
```

```
CREATE TABLE PROJECT (
  PROJ              CHAR(02)
  NOT NULL.
  NAME              CHAR(12) NOT NULL,
  DEPT              CHAR(03)
  NOT NULL WITH DEFAULT,
  MAJPROJ           CHAR(02) ) IN DATABASE SYSGROUP;
```

Figure 2. Result Tables:

SELECT * FROM EMP

EMP TABLE

NBR	LNAME	FNAME	DOB	HIREDATE	PERF	JOB	DEPT	PROJ
01	LOWE	BOB	1953-01-12	1985-10-12	4	PROG	FIN	01
02	SHIELD	BROOKE	1953-07-31	1987-07-01	3	MAN	MKT	01
03	MOORE	ROGER	1948-06-11	1986-06-02	1	DIR	MKT	04
04	EASTWOOD	CLINT	1941-02-21	1962-01-20	3	PROG	FIN	03
05	MOSTEL	ZERO	1961-11-15	1984-11-11	-	PRES	-	-
06	BURNS	GEORGE	1911-03-28	1949-09-01	2	SYS	FIN	01
07	O'NEIL	RYAN	1942-07-19	1960-10-21	3	DIR	ACC	05
08	MARVIN	LEE	1932-04-07	1956-11-76	2	VP	ACC	02
09	LANCASTER	BURT	1941-03-01	1979-12-92	1	MAN	R&D	02
10	BLAIR	LINDA	1954-11-13	1980-01-12	1	PROG	MKT	-

SELECT * FROM PROJ

PROJ TABLE

NBR	NAME	DEPT	MAJPROJ
01	PHASERS	MKT	-
02	SYSTEM X	MKT	03
03	SYSTEM R	FIN	-
04	LASERS	ACC	01
05	R*	FIN	02
06	NEW PROJ	R&D	03

Figure 3. Sample View Definition:

```
CREATE VIEW YOUNG_EMPLOYEES
( LAST_NAME, FIRST_NAME,
  DATE_OF_BIRTH, DEPT)
AS SELECT LNAME, FNAME, DOB, DEPT
FROM EMP
WHERE DOB > '1950-01-01'
```


(Continued From Page 17)

nition is conceptually represented in Figure 5. In fact, as will be demonstrated in a later example, the DB2 component that accomplishes the text/merge processing is more intelligent than the simple operation shown in figure 5 illustrates.

Notice from Figure 5 that the:

"Run-time" SELECT over-ride the VIEW's column definition internally to extract rows from EMP. However the column headers presented in SPUFI/QMF will reflect the view definition.

VIEW Definition's WHERE specifications were merged, in fact, ANDed together with the "run-time" SELECTs'

ORDER BY was appended to the composite statement from the run-time SELECT

What standard IBM processing activity does this text-merge operation remind you of? If you thought of merging run-stream JCL with stored procedures (PROCs) then we're on the same wave length. Just as in JCL/PROC execution the MVS reader-interpreter overrides and merges PROC symbolics and DD cards, RDS (the DB2 subcomponent responsible for view execution) overrides and merges run-time SQL SELECT clauses with the view definition stored in the DB2 System catalog.

RESTRICTIONS ON VIEWS

Many views are considered by DB2 to be "read-only", that is, only SELECT access is allowed against the view. Views in this category consist of views built on any of the following SQL constructs:

- Table joins
- DB2 Aggregate functions
- DB2 Scalar functions
- Group By/Having
- Distinct
- or the view references a read-only view in the FROM clause.

Figure 4. Results Table Built By Select Against View Definition:

SELECT * FROM YOUNG_EMPLOYEES			
LAST_NAME	FIRST_NAME	DATE_OF_BIRTH	DEPT
LOWE	BOB	1953-01-12	FIN
SHIELD	BROOKE	1953-07-31	MKT
MOSTEL	ZERO	1961-11-15	-
BLAIR	LINDA	1954-11-13	MKT

YOUNG_EMPLOYEES VIEW

```
CREATE VIEW YOUNG_EMPLOYEES
(LAST_NAME, FIRST_NAME,
DATE_OF_BIRTH, DEPARTMENT)
AS
SELECT LNAME, FNAME, DOB, DEPT
FROM EMP
WHERE DOB > '1950-01-01';
```

RUN-TIME SELECT

```
SELECT LAST_NAME, DEPT
FROM YOUNG_EMPLOYEES
WHERE DEPT IN ('FIN', 'MKT')
ORDER BY LNAME;
```

```
SELECT LNAME, FNAME
FROM EMP
WHERE DOB > '1950-01-01' AND
DEPT IN ('FIN', 'MKT')
ORDER BY LNAME;
```

Figure 5.

Composite Executable Statement And Results Table Generated by Select Against View Definition.

LAST_NAME	DEPT
BLAIR	MKT
LOWE	FIN
SHIELD	MKT

We should also mention here that the UNION and ORDER BY clauses are not allowed in the definition of a view.

Because of these restrictions, views provide only partial logical program/data independence. Full logical program/data independence provided by views awaits until all theoretically updateable views, in fact are updateable (i.e. views consisting of table joins are theoretical-

ly updateable, views which contain GROUP BY/HAVING are not theoretically updateable, etc.). This unfortunate "state-of-the-art" is an industry wide problem, and may be corrected in later releases of DB2. Now let's revisit our original view capabilities list, from page 16, and show examples for each of the capabilities.

(Continued On Page 19)

(Continued From Page 18)

1. Provide custom user/application data access:

NEED: Provide the Marketing department with a view of the Employee table composed of only Marketing department employees.

```
CREATE VIEW MKT_EMPL AS
SELECT * FROM EMP
WHERE DEPT = 'MKT'
```

NEED: Provide the general public SELECT access to specific Employee information, including certain columns. Restrict access to the President and Vice Presidents' data

```
CREATE VIEW EMPLOYEE_INFO AS
SELECT LNAME, FNAME, JOB, DEPT
FROM EMP
WHERE JOB NOT IN ('PRES','VP')
```

NEED: Provide a project level view of our employees. Include the project name, sponsoring department, Employee names, employee jobs and the employees' department. Rename the columns in this view to make them more meaningful

```
CREATE VIEW PROJECT_EMP
(PROJECT_NAME, SPONSORING_DEPT, EMPL_NAME, EMPL_JOB, EMPL_DEPT) AS
SELECT NAME, PROJ.DEPT, LNAME, JOB, EMP.DEPT
FROM EMP, PROJ
WHERE EMP.PROJ = PROJ.NBR
```

NEED: Display performance statistics for our employees broken down by department.

```
CREATE VIEW EMPL_STATS
(DEPT_NAME, MAX_PERF, AVG_PERF, MIN_PERF, COUNT_EMPS) AS
SELECT DEPT, MAX(PERF), AVG(PERF), MIN(PERF), COUNT(*)
FROM EMP
GROUP BY DEPT
```

NEED: Create a custom view for the personnel department consisting of employee last names concatenated with the first initial of their first names, the month they were hired and their department. If the department is null, replace the null with asterisks.

```
CREATE VIEW PERSONNEL
(EMPLOYEE, HIRE_MONTH, DEPARTMENT) AS
SELECT LNAME || SUBSTR(FNAME,1,1), MONTH(HIREDTE), VALUE(DEPT, '***')
FROM EMP
```

Select * From Personnel

EMPLOYEE		HIRE_MONTH	DEPARTMENT
=====		=====	=====
LOWE	R	01	FIN
SHIELD	B	07	MKT
MOORE	R	06	MKT
EASTWOOD	C	02	FIN
MOSTEL	Z	11	***
BURNS	G	03	FIN
O'NEIL	R	07	ACC
MARVIN	L	04	ACC
LANCASTER	B	03	R&D
BLAIR	L	11	MKT

Using the view mechanism to build these types of structures can free DB2 system-users from unnecessary and ongoing SQL coding effort. The view definitions are a one-time expenditure of energy, and the SQL coding-techniques used are entirely up to the view creator(s) discretion (i.e. who do you trust the most to make things efficient, and to get things straight?)

(Continued On Page 20)

(Continued From Page 19)

2. Provide logical program/data independence:

NEED: Most of a new IMS DC online application (250 programs) will access the EMP table but most of the DB2 data access is restricted to either employee specific information or department specific information. A small number of other application programs and a small but determined bunch of QMF users frequently access all the employee table columns. Split the Employee table vertically, creating an Employee info table and a Department info table. Do this with as little impact on the current production environment as possible.

2.a

```
CREATE TABLE EMP_INFO (
```

```
  NBR .....
  LNAME .....
  FNAME .....
  DOB .....
  HIREDTE ....
  PERF ....
  ....
```

```
CREATE TABLE DEPT_EMP_INFO (
```

```
  NBR .....
  DEPT .....
  JOB .....
  PROJ .....
  ....
```

```
CREATE VIEW EMP AS
```

```
  SELECT E.NBR, LNAME, FNAME, DOB, HIREDTE, PERF, DEPT, JOB, PROJ
  FROM EMP_INFO E, DEPT_EMP_INFO D
  WHERE E.NBR = D.NBR
```

NEED: Several renegade departments refuse to recognize the necessity of a corporate data model. They want their employees to see only the data belonging to their respective departments. Implement a single view to accomplish this. Each employees' TSO ID corresponds to their employee number.

2.b

```
CREATE VIEW DEPARTMENT_VIEW AS
```

```
  SELECT * FROM EMP
  WHERE DEPT IN
    (SELECT DEPT FROM EMP WHERE NBR = USER)
```

NEED: A new data model has been constructed and the EMP table must be "upgraded" to reflect new column naming standards. Also several columns of the EMP table must be deleted - due to new Entity-Attribute diagrams. Management doesn't want you to drop the table because of the impact on existing associated DB2 structures. Create the following view:

2.c

```
CREATE VIEW RESTRUCTURE_TABLE
```

```
  (EMPNO, LAST_NAME, FIRST_NAME, DATE_OF_BIRTH, DATE_OF_HIRE, EMP_PROJ) AS
  SELECT NBR, LNAME, FNAME, DOB, HIREDTE, PROJ
  FROM EMP
```

I think that extending logical data/program independence was in the backs of the minds of the originators of the view mechanism. Views are the central support structure for insulating applications from changes to DB2 base tables.

(Continued On Page 21)

(Continued From Page 20)

3. Provide an extra measure of data security and integrity:

The relational purists may wince at the prospect of degrading the lofty view to the level of Grant/Revoke statements. Never the less, many installations successfully use views as part of a comprehensive DB2 security and data access strategy.

NEED: Create a custom view for QMF end-user access composed of the employee last name, first name and department columns (i.e. While UPDATE access can be Granted and Revoked at the column level, SELECT access cannot. SELECT is granular only at the table level/view level).

3.a

```
CREATE VIEW COLM_SELECT_ACCESS AS
  SELECT LNAME, FNAME, DEPT
  FROM EMP
```

NEED: Create row level access to the Employee table so that Users can only retrieve rows corresponding to their specific departments. (Since building the view in 2.c management has been assigning employee numbers that do not correspond to TSO IDs)

3.b

```
CREATE TABLE EMP_ID (
  TSO_ID CHAR(08) NOT NULL,
  EMP_NBR INTEGER NOT NULL)
IN DATABASE SYSGROUP;
```

Populate this table with all employee/TSO_ID information.

```
INSERT INTO EMP_ID VALUES ('TSO001','01');
INSERT INTO EMP_ID VALUES ('TSO002','21');
INSERT INTO EMP_ID VALUES ('TSO003','56');
```

Finally, create a security view as follows:

```
CREATE VIEW EMP_DEPT AS
  SELECT * FROM EMP
  WHERE DEPT IN
    (SELECT DEPT FROM EMP, EMP_ID
     WHERE NBR = EMP_NBR AND
     TSO_ID = USER)
```

NEED: Create a "leveled" access to the Employee table based on a four part level scheme. Rows designated as level#1 are the most secure followed by level#2, and level#3 rows. Level#4 rows should be accessible by the general public. Users are assigned a security level, and should have access to all rows at a lower (higher number) level. Unlike the original employee number/TSO ID scheme discussed in figure 2.b, management has been assigning employee numbers that do not correspond to TSO IDs)

```
CREATE TABLE EMP_ID (
  TSO_ID CHAR(08) NOT NULL,
  EMP_NBR INTEGER NOT NULL,
  LEVEL# SMALLINT NOT NULL)
IN DATABASE SYSGROUP;
```

Populate this table with all employee/TSO_ID, Security level number information as follows:

```
INSERT INTO EMP_ID VALUES ('TSO001','01',2);
INSERT INTO EMP_ID VALUES ('TSO002','21',1);
INSERT INTO EMP_ID VALUES ('TSO003','56',4);
```

ALTER the EMP table. Add a LEVEL# column and update the column to reflect the security level of each particular row.

```
UPDATE EMP SET LEVEL# = 4;
UPDATE EMP SET LEVEL# = 3 WHERE PROJ = '02';
UPDATE EMP SET LEVEL# = 2 WHERE PROJ IN ('01','05','07');
UPDATE EMP SET LEVEL# = 1 WHERE JOB IN ('PRES','VP');
```

(Continued On Page 22)

(Continued From Page 21)

Finally, create the following leveled security view, and grant access to the EMP table through this view:

3.c

```
CREATE VIEW EMP_DEPT AS
SELECT * FROM EMP
WHERE DEPT IN
  (SELECT DEPT FROM EMP, EMP_ID
   WHERE NBR = EMP_NBR      AND
         TSO_ID = USER      AND
         EMP.LEVEL# >= EMP_ID.LEVEL# )
```

NEED: Support the following domain integrity rules for the EMP table:

3.d

- Employee numbers shall be greater than 0 and less than 1,000
- Valid birthdays are between 1910 and 1965
- Valid hiredates are between 1950 and the present
- Valid hiredates are at least 16 years later than the employees birthday.
- Valid departments include: ACC, FIN, R&D, MKT. All others are invalid including nulls (Note that I did not specify NOT NULL on my table definition. By imposing this condition with a view you can obtain additional flexibility)
- Valid jobs include: PROG, SYS, VP, DBA, PRES, MAN
- Valid projects include: 01, 02, 03, 04, 05, 06. (Note that you cannot generate a list of valid projects with a subselect because of DB2's restriction against updating through a view that references a different table in the subselect)

```
CREATE VIEW DOMAIN_INTEGRITY AS
SELECT * FROM EMP
WHERE NBR BETWEEN 0 AND 1000      AND
      YEAR(DOB) BETWEEN 1910 AND 1965      AND
      YEAR(HIREDATE) BETWEEN 1950 AND YEAR(CURRENT DATE) AND
      HIREDATE > DOB + 16 YEARS      AND
      DEPT IN ('ACC','FIN','R&D','MKT')      AND
      JOB IN ('PROG','SYS','VP','DBA','PRES','MAN')      AND
      PROJ IN ('01','02','03','04','05','06')
WITH CHECK OPTION
```

Note that it is the check option that forces DB2 to verify values that are Inserted and updated through the view. What this type of processing really accomplishes is to make better use of the System Catalog tables. Now, not only is your "metadata" (Entities and Attributes, user data and relationships among the data) stored in the catalog tables, but the business events and rules for the system are also stored in the DB2 catalog.

4. Provide DB2 system and application efficiency controls:

Here I'm not going to provide explicit examples of view usage. Just realize that poor SQL coding techniques can easily overwhelm the most careful and rigorous physical database design. And unfortunately:

- 1) Even well trained and technically competent application developers may not know the most efficient path through the current physical status of the database - including available indexes, key placement within indexes, column cardinality etc.
- 2) Often well trained and technically superior application developers are not available, and management will not commit to required advanced training for the programming staff
- 3) Business end-users seldom have the slightest concern for SQL efficiency coding principles

Some of the problems associated with poor coding techniques can be overcome through the use of views. You can create "copybook views" for application programs and business users that contain:

- Optimized joins
- Optimized predicate logic for "simple" queries
- Field element data access (so that even if your programmers and business-users are coding SELECT * they won't be SELECTing *)

Again, these are one time, or infrequent expenditures of energy that could potentially save considerable I/O and CPU processing.

(Continued On Page 23)

(Continued From Page 22)

5. Facilitate application development overcoming some of the compromises of IBM SQL:

NEED: We need to code a subselect that SELECTs all three columns of our primary key (NBR, LNAME, FNAME). SQL does not support SELECTing more than one column in a subselect. Further complicating the issue is the fact that LNAME and FNAME are CHAR columns while NBR is INTEGER.

5.a

```
CREATE VIEW SQL_SOLUTION
(PRIME_KEY, DOB, HIREDTE, DEPT, JOB, PERF, PROJ) AS
SELECT SUBSTR(DIGITS(NBR),4,3) || LNAME || FNAME, DOB,
HIREDTE, DEPT, JOB, PERF, PROJ
FROM EMP
```

Note that in the above SELECT clause we are concatenating numeric and character fields to form a three part "key" that is treated by SQL as a single column.

This allows us to execute the following statement:

```
SELECT ... FROM SQL_SOLUTION WHERE PRIME_KEY IN
(SELECT PRIME_KEY FROM SQL_SOLUTION WHERE ...)
```

NEED: Obtain the average maximum and minimum performance evaluations by department. Do not use QMF. (Note: IBM SQL does not currently support taking an aggregate function of an aggregate function AVG(MAX(PERF)) etc.)

5.b

```
CREATE VIEW FUNC_ON_FUNC
(DEPT, MAX_PERF, MIN_PERF) AS
SELECT DEPT, MAX(PERF), MIN(PERF)
FROM EMP
GROUP BY DEPT
```

Now you may:

```
SELECT DEPT, AVG(MAX_PERF), AVG(MIN_PERF)
FROM FUNC_ON_FUNC
GROUP BY DEPT
```

In the last two examples, we see that the DB2 component that materializes views (Relational Data System or RDS), is actually more intelligent than a simple language text/merge routine. If it were not, when the FUNC_ON_FUNC and SQL_SOLUTION views with the run-time SELECTs against those views were merged, an invalid SQL statement would result. This statement would not be executable.

(Continued On Page 24)

(Continued From Page 23)

CONSIDERATIONS FOR USING VIEWS

Performance Penalty

While there is a slight execution penalty for using views (one extra system catalog read to obtain the operational view text) this is usually a "micro" price compared to the "macro" costs surrounding the problems that views can address.

View Administration

A more perplexing problem facing the extensive use of DB2 Views might be one of view administration. Using views can add a factor of complexity to existing data administration policies and procedures. Also, the traditional political/philosophical issues can arise concerning who "owns" the view definitions, and who is responsible to maintain them (I am thinking here particularly of "applications oriented" views). As an aside, you should realize the while DBADM authority can create views for other IDs in databases, only SYSADM can DROP views on other IDs. This may cut against the grain of your current system authorization and privilege administration.

Views and the DB2 LOAD Utility

A problem that may surface in trying to accomplish too much through the use of views involves the DB2 LOAD Utility. LOAD (like most other DB2 utilities) attaches to the DB2 Data Management services at a level below the component that understands "relational theory" (Relational Data System - or RDS). This means that data integrity controls implemented through views will be defeated during the LOAD operation, and techniques covered in this article like supporting domain integrity through views (example 3.d) will not work for LOAD.

IN CONCLUSION

Views allow users to focus on the particular data that interests them, and on the particular tasks for which they're responsible. While views certainly aren't a panacea for all DB2/SQL application development woes, judicious use of this technique can expedite many concerns, particularly in the areas of supporting logical program/data independence, data security/integrity controls, and DB2 application/system efficiency. I hope the information in this article has been useful to you, and I look forward to hearing from you regarding the use of views in DB2 development efforts.

Jonathan Sayles is the director of Educational Services for The Systems Group, Inc., a technical training and consulting firm that specializes in DB2 and relational database systems. He is the author of several books on DB2 available through Q.E.D. Information Sciences, Inc., of Wellesley Mass., and the developer of a Computer Assisted Instruction course on SQL, also available through Q.E.D. Information Sciences. Mr. Sayles has spent the last three years teaching DB2 and working on DB2 application systems and DB2 related C.A.S.E. tools.